Merkle Aidrop Audit

Issue Findings

Design thinking

[01] merkle tree leaf structure re-design

A Merkle tree theoretically allows same user's address and corresponding amount to be added to the same Merkle multiple times (refer this test case)

For example, it's possible to reward user A with 100 tokens and then, after ‘updateMerkleRoot’, reward user A again with 200 tokens. However, if according to the logic

require(_claimedTokens[receiver][id] < amount, "Insufficient tokens available to claim!");

for the second reward, user A would never be able to claim it.

Considering this, I changed the structure of the leaf. I've added a global unique index for each leaf. Even if it's the same user, in different rewards, the corresponding index will be unique each time. Uniswap also use a similar design.

also, I've transformed:

mapping(address tokenReceiver => mapping(string projectId => uint64 claimedTokens)) private _claimedTokens;

to:

mapping(address tokenReceiver => mapping(string projectId => mapping(uint256 index => bool claimed))) private _claimed;

Not only I added an index level in the mapping, I also changed the claimed amount (`uint64`) to a boolean value.

[02] set Max project name length for project name

like this

```
require(bytes(projectName).length < 64, "Invalid project length")
```

[03] ProjectId should set to uint type

I'm not sure why the project id is set to string type. If it is as a unique identifier, i think we can set it as uint64 or uint256

[04] Struct Layout optimization

consider the project name length is dynamic, so i put it in the end of this structure,

additionally, according to the gas report from hardhat, i found different layout for the Project, have different effect for setProject, updateProject, issueToken functions

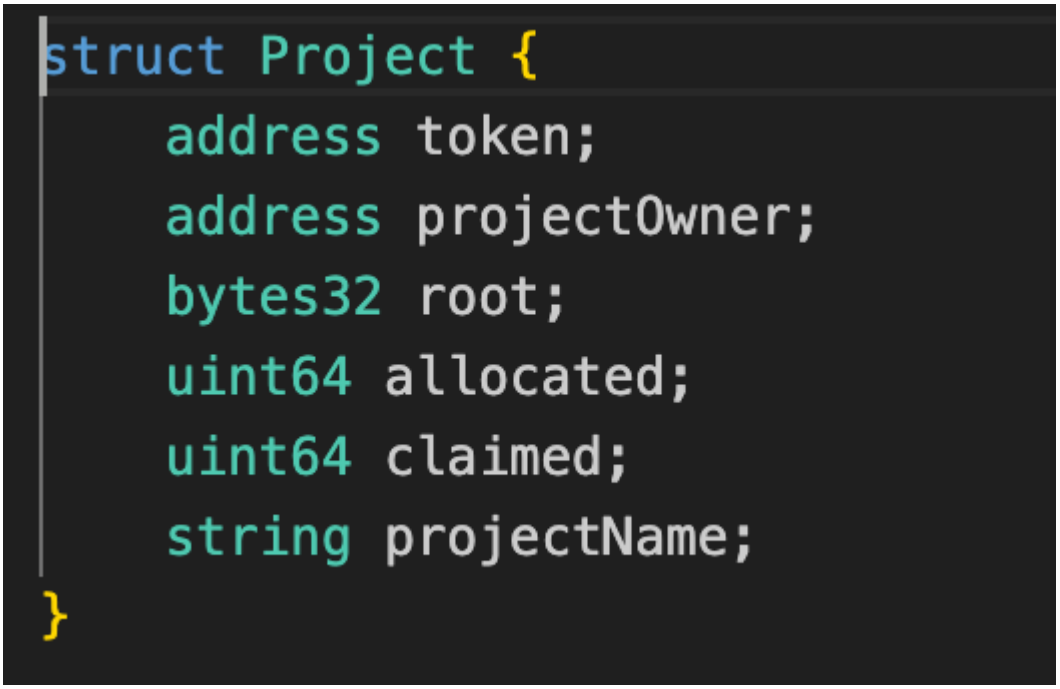
according to the usage scenarios, issuetokens should be called most frequently, so I chose the layout which can make issueTokens gas cost minimum

refer this blog:https://docs.uniswap.org/blog/intro-to-gas-optimization

from

```
struct Project {
    address token;
    bytes32 root;
    string projectName;
    address projectOwner;
    uint64 allocated;
    uint64 claimed;
}
```

to



```
struct Project {
    address token;
    address projectOwner;
    bytes32 root;
    uint64 allocated;
    uint64 claimed;
    string projectName;
}
```

constructor

[01] constructor gas optimize

Because the length of added tokens is always limited and far from reaching the overflow standard, using unchecked to wrap the i++ operation saves gas.

From

```
for (uint256 i = 0; i < tokens.length; i++) {
    _tokens[tokens[i]] = true;
}
```

To

```
for (uint256 i = 0; i < tokens.length; ) {
    _tokens[tokens[i]] = true;
    unchecked {
        i++;
    }
}
```

issueTokens function

[01] Error message misunderstanding in issueTokens

From

```
require(token != address(0), "Project already exists!");
```

To

```
require(token != address(0), "Project not exists!");
```

[02] Redundant code in issueTokens

redundant code

```
require(
    allocated > claimed,
    "All allocated tokens have already been claimed! Please contact the project owner."
);
```

actualy, below code already check it

if (allocated > claimed) {
    …
} else {
        revert("No tokens available to claim!");
}

[03] Wrong approve/Transferfrom in issueTokens

below code in issues token is wrong, actually, user issue tokens, the token should be transfe from smart contract to user, so iit don't to be approved, also don't need to be transferFrom, just call 'safeTransfer'

From

```
IERC20(token).approve(address(this), availableClaimedTokens);
IERC20(token).transferFrom(address(this), receiver, availableClaimedTokens);
```

To

```
IERC20(token).safeTransfer(receiver, amount);
```

Also need to note that the transfer operation is always executed after the update storage operation.

reclaimTokens function

[01] gas optimze for reclaimTokens

Try to access local memory variables instead of storage

```
address token = _projects[id].token;
address projectOwner = _projects[id].projectOwner;
```

replace

```
require(address(_projects[id].token) != address(0), "Project does not exist!");
require(msg.sender == _projects[id].projectOwner, "Caller is not project owner!");
```

to

```
require(token != address(0), "Project does not exists!");
require(msg.sender == projectOwner, "Caller is not project owner!");
```

[02] Wrong approve/Transferfrom in reclaimTokens

From

```
IERC20(_projects[id].token).approve(address(this), reclaimAmount);
IERC20(_projects[id].token).transferFrom(address(this), _projects[id].projectOwner, reclaimAmount);
```

to

```
IERC20(token).safeTransfer(projectOwner, reclaimAmount);
```

setProject function

[01] zero check for allocated amount in setProject

should add zero check for allocated amount

```
require(allocated > 0, "Allocated amount should greater than 0");
```

[02] Use SafeTransferFrom for setProject

from

```
    IERC20(token).transferFrom(projectOwner, address(this), allocated);
}
```

to

```
IERC20(token).safeTransferFrom(projectOwner, address(this), allocated);
```

updateProject function

[01] gas optimze for updateProject

claimed amount only be used once, so it don't need to be store at a local variable

from

```
uint64 claimed = _projects[id].claimed;
require(claimed == 0, "Project has started claimed");
```

to

```
require(_projects[id].claimed == 0, "Project has started claimed");
```

[02] Replace Transfer by SafeTransfer and put them behind update action

From

```
IERC20(oldTokenAddress).transfer(projectOwner, oldAllocated);

_projects[id] = Project(token, root, projectName, projectOwner, allocated, 0);

IERC20(token).transferFrom(projectOwner, address(this), allocated);
```

to

```
_projects[id] = Project(token, projectOwner, allocated, 0, root, projectName);

IERC20(oldTokenAddress).safeTransfer(projectOwner, oldAllocated);
IERC20(token).safeTransferFrom(projectOwner, address(this), allocated);
```

deposit function

[01] Replace TransferFrom by SafeTransferFrom

from

```
IERC20(token).transferFrom(msg.sender, address(this), amount);
```

to

```
IERC20(_projects[id].token).safeTransferFrom(msg.sender, address(this), amount);
```

Add events for external interface

event should be added to each external interface. I only added events to issuetokens and reclaimTokens.

```
event TokenIssued(address indexed account, string id, uint256 index, uint64 amount);
event TokenReclaimed(address indexed owner, string id, uint64 amount);
```

conclusions

actually, there are still many areas for improvement in this smart contract. due to time constraints, I have not optimized it to the most perfect state.